Numerical Methods for Computational Science and Engineering

Fall Semester 2018

Prof. Rima Alaifari, SAM, ETH Zurich

1. Computing with vectors & matrices

1.1. Numerics & Error Analysis

Real-world quantities: $\mathbb{R}$ / $\mathbb{C}$

Computers can't compute properly in $\mathbb{R}$

↳ Set of machine numbers $\mathbb{M}$: finite & discrete

$$\mathbb{M} \subsetneq \mathbb{R}$$

$\mathbb{M}$ is not closed under basic arithmetic operations

$$op \in \{'*', '/', '+', '-'\}$$

$$op: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

$$\mathbb{M} \times \mathbb{M} \not\to \mathbb{M}$$

Code Snippet 1. 1: *Machine Arithmetic Example*

```cpp
#include <iostream>
using namespace std;

double a = 1.0;
double b = a/9.0;
if(a==b*9.0) cout << "They are equal";
else cout << "They are not equal";
/*
```

will print: not equal

$$\frac{1}{9} = 0.\dot{1}$$

Instead of == queries: ask for approximate equality some given tolerance

Code Snippet 1. 2: *Machine Arithmetic Example*

```
#include <limits>
#include <iostream>
using namespace std;

double a = 1.0;
double b = a/9.0;
if(fabs(a-b*9.0)<numeric_limits<double>::epsilon)
    cout << "They are equal";
else cout << "They are not equal";
/*
```

machine precision

→ introduces tradeoff between tolerance & accuracy

Fixed Point Representation:

fixed decimal point → most straight forward way to store numbers

$k, l \in \mathbb{N}$ range 10^{-k} to 10^{l}

$k+l+1$ digits k of which appear after decimal point

Pro: arithmetic operations: almost as if working with integers

$$a+b = (a \cdot 10^k + b \cdot 10^k) \cdot 10^{-k}$$

Con: precision issue

e.g. $k=1$: $0.1 * 0.1 \; (= 0.01)$
$\cong 0$ (truncation)

Used in systems that favor time over accuracy (e.g. GPU systems)

"Default": Floating Point Representation

In applications useful with frequent change of scales → requires unified representation

Floating Point Representation:

$$\pm\ \underbrace{0.73125}_{\text{digits of mantissa}} \cdot \underbrace{10}_{\text{base}}^{\overbrace{12}^{\text{exponent}}}$$

Definition 1.1.1 (Machine numbers/floating point numbers). Given:

- Basis $B \in \mathbb{N} \setminus \{1\}$
- exponent range $\{e_{\min}, \ldots, e_{\max}\}, e_{\min}, e_{\max} \in \mathbb{Z}$ and $e_{\min} < e_{\max}$
- number $m \in \mathbb{N}$ of digits (for mantissa)

the corresponding set of *machine numbers* is:

$$\mathbb{M} := \{d \cdot B^E : d = i \cdot B^{-m}, i = B^{m-1}, \ldots, B^m - 1, E \in \{e_{\min}, \ldots, e_{\max}\}\}$$

$$x = \pm\ 0.\boxed{\ \ \ \ \ \ \ \ \ \ \ \ } \cdot B^{\boxed{\ \ \ \ \ \ }}$$

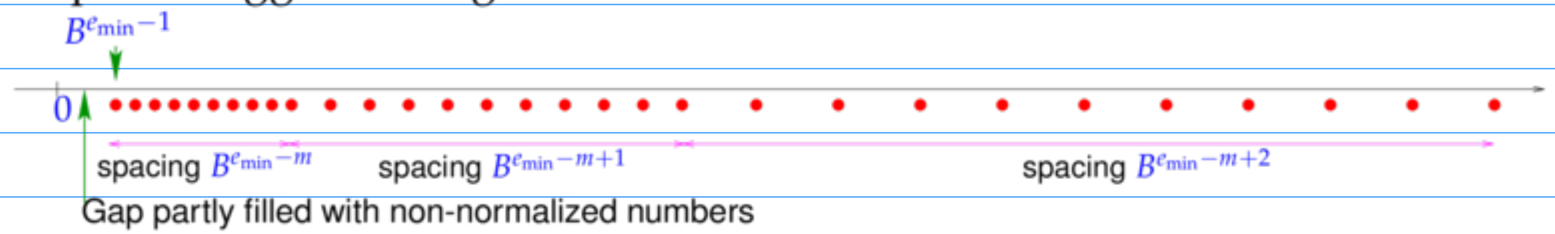
first digit $\neq 0$

Machine numbers $\mathbb{M}$: standard for floating point repr.: IEEE754

most common: binary32
binary64

Remark. Machine numbers are not evenly spaced!

Gaps are bigger for large number:



Recall: $op : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$

On computer: $\widetilde{op} : \mathbb{M} \times \mathbb{M} \to \mathbb{M}$

implementation: $\widetilde{op} = rd \circ op$

Definition 1.1.4 (Correct rounding). Correct rounding ("rounding up") is given by the function

$$\mathrm{rd}: \begin{cases} \mathbb{R} \to & \mathbb{M} \\ x \mapsto & \max \operatorname{argmin}_{\tilde{x}\in\mathbb{M}} |x-\tilde{x}| \,. \end{cases}$$

Definition 1.1.2 (Absolute and relative error). Let $\tilde{x} \in \mathbb{K}$ be an approximation of $x \in \mathbb{K}$. Then its *absolute error* is given by

$$\epsilon_{\text{abs}} := |x-\tilde{x}| \,,$$

and its *relative error* is defined as

$$\epsilon_{\text{rel}} := \frac{|x-\tilde{x}|}{|x|} \,.$$

Approximation $\tilde{x}$ of x has $\ell \in \mathbb{N}_0$ correct digits if

$$\varepsilon_{rel} := \frac{|x-\tilde{x}|}{|x|} \leq 10^{-\ell}$$

Maximal relative error of rounding:

$$\mathrm{EPS} = \max_{x\in\mathbb{R}} \frac{|\mathrm{rd}(x)-x|}{|x|}$$

machine precision

Assumption 1.1.1 ("Axiom" of roundoff analysis). *There is a small positive number* EPS, *the* machine precision, *such that, for the elementary arithmetic operations* $\star \in \{+,-,\cdot,/\}$ *and "hard-wired" functions* * $f \in \{\exp, \sin, \cos, \log, \ldots\}$, *the following holds*

$$x \,\tilde{\star}\, y = (x \star y)(1+\delta) \quad , \quad \tilde{f}(x) = f(x)(1+\delta) \quad \forall x,y \in \mathbb{M}\,,$$

Alternative way to understand EPS:

smallest positive number s.t.

$1 \,\tilde{+}\, \mathrm{EPS} \neq 1$. (note: e.g. $10 \,\tilde{+}\, \mathrm{EPS} = 10$)

Note: other sources of errors exist:

measurement error
modeling error
discretization error
etc.

Note: Rel. & abs. error: in general not computable

(don't know true solution!)

1. Worst case estimates

2. Compute backward error

Suppose we want to solve for

$Ax = b$ $\qquad$ x_{ex} (true sol.)

compute x_{app}

$b_{app} := Ax_{app}$

compare b_{app} to b

forward error: $\| x_{ex} - x_{app} \|$

backward error: $\| b - b_{app} \|$ ⟸ computable!

In practice: stop when $Ax_{app} - b$ small

<u>However</u>:

small backward error

$\not\Rightarrow$ small forward error

x_{app} can be far off from x_{ex}.

Condition number:

Example of matrix A:

$$\text{cond}(A) = \frac{\sigma_{max}}{\sigma_{min}}$$ ratio of largest & smallest singular value

Recall toy example : 3x3 matrix

$\|b-b^\delta\| = 0.007$ $\qquad \|x-x^\delta\|$ huge

condition number of A: $\underline{\underline{10^5}}$!

1.2 Fundamentals

1.2.1. Notation

$$A := \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{K}^{m,n}$$

$\mathbb{K}$: $\mathbb{R}/\mathbb{C}$

$(A)_{i,j} = a_{ij}$ $\qquad i \in \{1, \dots, m\}$

$\qquad j \in \{1, \dots, n\}$

$a_{i,:} = (A)_{i,:}$ $\qquad$ i-th row

$a_{:,j} = (A)_{:,j}$ $\qquad$ j-th column

$(a)_{\substack{i=k,\dots,l \\ j=r,\dots,s}} = (A)_{k:l,r:s}$ $\qquad 1 \le k \le l \le m$

$\qquad 1 \le r \le s \le n$

(submatrix)

A^T $\qquad$ transpose of A

A^H $\qquad$ adjoint of A $\qquad A^H = \begin{bmatrix} \overline{a_{11}} & \cdots & \overline{a_{m1}} \\ \vdots & & \vdots \\ \overline{a_{1n}} & \cdots & \overline{a_{mn}} \end{bmatrix}$

$A \in \mathbb{R}^{m,n}$: $A^H = A^T$

symmetric: $A^T = A$, Hermitian: $A^H = A$

Definition (s.p.d. matrix)

The matrix $A \in \mathbb{K}^{n,n}$, $n \in \mathbb{N}$, is symmetric (or Hermitian) (semi-) positive definite (s.p.d.) if

$A = A^H$ and

$\forall x \in \mathbb{K}^n$: $x^H A x \in \mathbb{R}$ and

$x^H A x > 0 \iff x \neq 0$
$(\geq)$

A matrix $A \in \mathbb{K}^{n,n}$ is symm. (semi-) pos. def. if & only if all its eigenvalues are positive (non-negative).

1.3. Libraries

1.3.1 EIGEN

header-only C++ library for numerical computations

provides
- data structures
- (standard) operations (on matrices / vectors)

fundamental data type: matrix

$$\text{Matrix}\langle \underset{\substack{\text{type of coeffs} \\ \text{(double, long, ...)}}}{\text{typename scalar}},\ \underbrace{\text{int nrows, int ncols}}_{\text{fixed} \quad \text{dynamic}} \rangle$$

Example: Matrix<double, Dynamic, Dynamic>

→ size not known at compile time but treated as runtime variable

Convenience typedefs:

MatrixXd — X ↑ dynamic, d ← double

Matrix3f — 3 ↑ 3x3 matrix, f ← float

Matrix3d x; 3x3 matrix with array of uninitialized coeffs

MatrixXf y; dynamic size current size 0-by-0

MatrixXf y(6,9);
VectorXd x(12);
x.resize(5);

dynamic size array of coeffs. allocated with given size

Code Snippet 1. 9: Initializing special matrices in EIGEN

```
#include <Eigen/Dense >
// Just allocate space for matrix, no initialisation
Eigen::MatrixXd A(rows,cols);
// Zero matrix.  Similar to matlab command zeros(rows, cols);
Eigen::MatrixXd B = MatrixXd::Zero(rows, cols);
// Ones matrix.  Similar to matlab command ones(rows, cols);
Eigen::MatrixXd C = MatrixXd::Ones(rows, cols);
// Matrix with all entries same as value.
Eigen::MatrixXd D = MatrixXd::Constant(rows, cols, value);
// Random matrix, entries uniformly distributed in [0,1]
Eigen::MatrixXd E = MatrixXd::Random(rows, cols);
// (Generalized) identity matrix, 1 on main diagonal
Eigen::MatrixXd I = MatrixXd::Identity(rows,cols);
std::cout << "size of A = (" << A.rows() << ',' << A.cols() <<
    ')' << std::endl;
```

1.3.2. Dense Matrix Storage Formats

$A \in \mathbb{K}^{m,n}$: stored as array of length $m \cdot n$

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Row major (C-arrays, bitmaps, Python):

`A_arr`	1	2	3	4	5	6	7	8	9

Column major (Fortran, MATLAB, EIGEN):

`A_arr`	1	4	7	2	5	8	3	6	9

Indices start at 0!

To access coeffs $A(\text{int } a, \text{ int } b)$

$v(\text{int } a)$

$A(4) = 5$

default: column major (in Eigen)

$\rightarrow$ can be changed

```
// Template parameter ColMajor selects column major data layout
Matrix<double,Dynamic,Dynamic,ColMajor> mcm(nrows,ncols);
// Template parameter RowMajor selects row major data layout
Matrix<double,Dynamic,Dynamic,RowMajor> mrm(nrows,ncols);
```

Data storage format impacts runtime

Example: row / column access of a matrix stored in column major

significantly more run time as matrix size increases

1.4. Computational Effort

computational effort $\hat{=}$ number of elementary operations in a run

Computational effort $\propto$ runtime

1.4.1. Asymptotic complexity

How does the computational effort <u>scale</u> with the problem size?

→ comparison of algorithms & their performance

Definition 1.5.1 ((Asymptotic) complexity)**.** The asymptotic complexity of an algorithm characterises the worst-case dependence of its computational effort on one or more problem size parameter(s) when these tend to ∞.

Typical parameter: dimension of input vector, matrix

Worst case analysis: What is the maximal effort over the set of all admissible inputs?

Landau Θ - notation:

$$f(n) = \Theta(g(n)) \qquad f, g : \mathbb{N} \to \mathbb{R}$$

if $\exists C > 0 \quad n_* \in \mathbb{N}$ s.t.

$$\forall n \geq n_* : \quad f(n) \leq C \cdot g(n).$$

Asymptotic complexity predicts dependence of runtime on the size of the problem.

Ex.: $\quad \mathrm{cost}(n) = \Theta(n^2)$

<u>Example</u>: Conjecture $\quad t_i \approx C \cdot n_i^{\alpha}$

runtimes $\qquad$ problem sizes

$i = 1, \dots, N$

log-log-plot: $\log t_i \approx \log C + \alpha \log n_i$

data points (t_i, n_i) lie on a straight line with slope α.

1.4.2. Cost of basic operations